Exercice 1 : utilisation de la commande print

Affectez les variables temps et distance par les valeurs 6.892 et 19.7.
Calculez et affichez la valeur de la vitesse. V= ?
Améliorez l'affichage en formatant avec un chiffre après le point décimal.

```
# Créé par Claude, le 17/09/2025 en Python 3.7
# -*- coding: UTF-8 -*-
"""La fonction print()."""

# programme principal -----------------------------------------------
## affichage simple :
temps = 13.784
distance = 39.4
vitesse = distance/temps
print("vitesse =", vitesse)
## affichage formate :
print("{}".format("-"*28)) # ligne de séparation
print("\n vitesse = {:.2f} m/s".format(vitesse)) # arrondi a 2 chiffres
```

```
vitesse = 2.8583865351131745
----------------------------

 vitesse = 2.86 m/s
>>>
```

Exercice 2 : saisie au clavier (input)

Saisissez un flottant. S'il est positif ou nul, affichez sa racine, sinon affichez un message d'erreur.

```
"""Les instructions de choix."""
# import de la biblio math pour la racine carrée
from math import sqrt
# programme principal ------------------------------
x = float(input("donnez x ? "))
if x >= 0:
    y = sqrt(x)
    print("La racine de {:.2f} est {:.3f}".format(x, y))
else:
    print("On ne peut pas prendre la racine d'un reel negatf !")

print("\n A bientôt")
```

```
donnez x ? 24.6
La racine de 24.60 est 4.960

 A bientôt
>>>
```

Exercice 3 : input()

On désire sécuriser une enceinte pressurisée.
On se fixe une pression seuil et un volume seuil : **pSeuil = 2.3, vSeuil = 7.41**.
On demande de saisir la pression et le volume courant de l'enceinte et d'écrire un script qui simule le comportement suivant :
– si le volume *et* la pression sont supérieurs aux seuils : arrêt immédiat ;
– si seule la pression est supérieure à la pression seuil : demander d'augmenter le volume de l'enceinte ;
– si seul le volume est supérieur au volume seuil : demander de diminuer le volume de l'enceinte ;
– sinon déclarer que « tout va bien ».
Ce comportement sera implémenté par une alternative multiple.

```
# programme principal ------------------------------------------------
p_seuil, v_seuil = 2.3, 7.41
print("Seuil pression :", p_seuil, "\tSeuil volume ;", v_seuil, "\n")
pression = float(input("Pression courante = "))
volume = float(input("Volume courant = "))
if (pression > p_seuil) and (volume > v_seuil):
    print("\t pression ET volume elevés. ALARME Stoppez !")
elif pression > p_seuil:
    print("\t Il faut augmenter le volume")
elif volume > v_seuil:
    print("\t Vous pouvez diminuer le volume")
else:
    print("\t Tout va bien !")
```

```
Seuil pression : 2.3    Seuil volume ; 7.41

Pression courante = 2
Volume courant = 8
     Vous pouvez diminuer le volume
```

```
Pression courante = 5
Volume courant = 8
     pression ET volume elevés. ALARME Stoppez !
>>>
```

```
Pression courante = 3
Volume courant = 5
     Il faut augmenter le volume
```

```
Pression courante = 2
Volume courant = 6
     Tout va bien !
```

Ex 3bis :Un gardien de phare va aux toilettes cinq fois par jour or les WC sont au rez-de-chaussée. . .
Écrire une procédure (donc sans retour) ***hauteurparcourue*** qui reçoit deux paramètres le nombre de marches du phare et la hauteur de chaque marche (en cm), et qui affiche :
`Pour x marches de y cm, il parcourt z.zz m par semaine`.
On n'oubliera pas :
– qu'une semaine comporte 7 jours ;
– qu'une fois en bas, le gardien doit remonter ;
– que le résultat est à exprimer en m.

```
Combien de marches ? 100
Hauteur d'une marche (cm) ? 20
100 marches de 20 cm. Il parcourt 1400.00 m par semaine.
>>>
```

```
# Créé par claude, le 06/10/2025 en Python 3.7
# Metre parcouru par gardien
"""Gardien de phare."""
# fonctions
def hauteurParcourue(nb, h):
    print("{} marches de {} cm. Il parcourt {:.2f} m par semaine.".format(nb,h, nb*h*2*5*7/100.0))

# programme principal ------------------------------------------------
nb_marches = int(input("Combien de marches ? "))
hauteur_marche = int(input("Hauteur d'une marche (cm) ? "))
hauteurParcourue(nb_marches, hauteur_marche)

```

Exercice 4 : range()

Affichez les entiers de 0 à 15 non compris, de trois en trois, en utilisant une boucle for et l'instruction range().

```
# Créé par Claude, le 17/09/2025 en Python 3.7
# -*- coding: UTF-8 -*-
"""Instruction ' range et for '."""

# programme principal ----------------------------------
print(" Instruction <range> ".center(40, '-'), "\n")
print(range(5))
print(range(3, 10))
print(range(0, 10, 2))
print("\n"+" <range> dans un <for> ".center(40, '-'), "\n")
for i in range(0, 15, 3):
    print(i, end=" ")
print()
```

```
--------- Instruction <range> ----------


-------- <range> dans un <for> ---------

0 3 6 9 12
>>> ----
```

Exercice 5 : range()

Utilisez l'instruction break pour interrompre une boucle for d'affichage des entiers de 1 à 10 compris, lorsque la variable de boucle vaut 5.

```
# Créé par Claude, le 17/09/2025 en Python 3.7
# -*- coding: UTF-8 -*-
"""Instruction 'break'."""

# programme principal -------------------------
for i in range(1, 11):
    if i == 5:
        print("arrêt")
        break
    print(i, end=" ")
print()
```

```
1 2 3 4 arrêt

>>>
```

5-1. Utilisez l'instruction continue pour modifier une boucle for d'affichage de tous entiers de 1 à 10 compris, sauf lorsque la variable de boucle vaut 5.

```
# Créé par Claude, le 17/09/2025 en Python 3.7
# -*- coding: UTF-8 -*-
"""Instruction 'continue'."""

# programme principal -------------------------
for i in range(1, 11):
    if i == 5:
        continue
    print(i, end=" ")
print()
```

```
1 2 3 4 6 7 8 9 10
>>>
```

5-2 . Utilisez une *exception* pour calculer, dans une boucle évoluant de -3 à 3 compris, la valeur de sin(*x*)/*x*.

```
# Créé par Claude, le 17/09/2025 en Python 3.7
# -*- coding: UTF-8 -*-
"""Instruction 'try except'."""

# import
from math import sin
# programme principal ------------------------------
for x in range(-3, 4): # -3 -2 -1 0 1 2 3
    try:
        print("{:.3f}".format(sin(x)/x), end=" ")
    except:
        print("{:.3f}".format(float(1)), end=" ")
print()
```

```
0.047 0.455 0.841 1.000 0.841 0.455 0.047
>>>
```

Calculer la factorielle d'un nombre

```
# Créé par claude, le 30/09/2025 en Python 3.7
nombre = 6
resultat = 1
for i in range(1, nombre + 1):
    resultat *= i
print(f"La factorielle de {nombre} est :", resultat)
```

Exercice 6 : fonction()

Écrire une fonction cube qui retourne le cube de son argument.

Écrire une fonction volumeSphere qui calcule le volume d'une sphère de rayon r fourni en argument et qui utilise la fonction cube.
Tester la fonction volumeSphere par un appel dans le programme principal.

```
# Créé par Claude, le 17/09/2025 en Python 3.7
# -*- coding: UTF-8 -*-
"""Instruction ' range et for '."""

# import
from math import pi
# fonctions
def cube(x):
    """Calcule le cube de l'argument."""
    return x**3
def volumeSphere(r):
    """Calcule le volume d'une sphere de rayon <r>."""
    return 4 * pi * cube(r) / 3
# programme principal ------------------------------------------------
rayon = float(input("Rayon : "))
print("\nVolume de la sphere de rayon {:.1f} : {:.3f}".format(rayon, volumeSphere(rayon)))
```

```
Rayon : 5

Volume de la sphere de rayon 5.0 : 523.599
```

Exercice 7 : listes

Définir la liste : liste =[17, 38, 10, 25, 72], puis effectuez les actions suivantes :

- triez et affichez la liste ;
- ajoutez l'élément 12 à la liste et affichez la liste ;
- inversez et affichez la liste ;
- affichez l'indice de l'élément 17 ;
- enlevez l'élément 38 et affichez la liste ;
- affichez la sous-liste du 2ème au 3ème élément ;
- affichez la sous-liste du début au 2ème élément ;
- affichez la sous-liste du 3ème élément à la fin de la liste ;
- affichez la sous-liste complète de la liste ;
- affichez le dernier élément en utilisant un indiçage négatif.

Bien remarquer que certaines méthodes de liste ne retournent rien.

```
# Créé par Claude, le 17/09/2025 en Python 3.7
# -*- coding: UTF-8 -*-
"""Listes et fonctions."""

# programme principal ----------------------------
nombres = [17, 38, 10, 25, 72]
print(" Liste initiale ".center(50, '-'))
print(nombres, '\n')
#rien = input('"Entree"')
print(" Tri ".center(50, '-'))
nombres.sort()
print(nombres, '\n')
#rien = input('"Entree"')
print(" Ajout d'un element ".center(50, '-'))
nombres.append(12)
print(nombres, '\n')
#rien = input('"Entree"')
print(" Retournement ".center(50, '-'))
nombres.reverse()
print(nombres, '\n')
#rien = input('"Entree"')
print(" Indice d'un element ".center(50, '-'))
print(nombres.index(17), '\n')
#rien = input('"Entree"')
print(" Retrait d'un element ".center(50, '-'))
nombres.remove(38)
print(nombres, '\n')
#rien = input('"Entree"')
```

```
---------------- Liste initiale ----------------
[17, 38, 10, 25, 72]

---------------------- Tri -----------------------
[10, 17, 25, 38, 72]

--------------- Ajout d'un element ---------------
[10, 17, 25, 38, 72, 12]

------------------ Retournement ------------------
[12, 72, 38, 25, 17, 10]

-------------- Indice d'un element ---------------
4

-------------- Retrait d'un element --------------
[12, 72, 25, 17, 10]
```

```
#rien = input('"Entree"')
print(" Indicage ".center(50, '-'))
print("nombres[1:3] =", nombres[1:3])
print("nombres[:2] =", nombres[:2])
print("nombres[2:] =", nombres[2:])
print("nombres[:] =", nombres[:])
print("nombres[-1] =", nombres[-1])
```

```
--------------------- Indicage ---------------------
nombres[1:3] = [72, 25]
nombres[:2] = [12, 72]
nombres[2:] = [25, 17, 10]
nombres[:] = [12, 72, 25, 17, 10]
nombres[-1] = 10
>>>
```

Exercice 8 : module et import

```
# -*- coding: UTF-8 -*-
"""Module de fonctions pour les nombres parfaits et chanceux."""
# fichier : parfait_chanceux_m.py
# fonctions
def somDiv(n):
    """Retourne la somme des diviseurs propres de <n>."""
    somme = 1
    for i in range(2, n//2+1):
        if n % i == 0:
            somme += i
    return somme
def estParfait(n):
    """Teste si <n> est parfait."""
    return True if somDiv(n) == n else False
def estPremier(n):
    """Teste si <n> est premier."""
    return True if somDiv(n) == 1 else False
def estChanceux(n):
    """Teste si <n> est chanceux."""
    for i in range(0, n-1):
        if not estPremier(n + i + i**2):
            return False
        return True
# Auto-test ----------------------------------------------------------
if __name__=='__main__':
    print("Somme des diviseurs propres de 12 : ",somDiv(12))
    print("6 est-il parfait : ",estParfait(6))
    print("31 est-il premier : ",estPremier(31))
    print("11 est-il chanceux : ",estChanceux(11))
```

Tab en trop

```
Somme des diviseurs propres de 12 :  16
6 est-il parfait :  True
31 est-il premier :  True
11 est-il chanceux :  True
```

Exercice 9 :

```
# Créé par Claude, le 17/09/2025 en Python 3.7
# imports
from parfait_chanceux_m import estParfait, estChanceux
# programme principal ------------------------------------------------
parfaits, chanceux = [], []
intervalle = range(2, 1001)
for i in intervalle:
    if estParfait(i):
        parfaits.append(i)
    if estChanceux(i):
        chanceux.append(i)
msg = " Nombres remarquables dans [2 .. 1000] ".center(70, '-')
msg += "\n\nParfaits :\t{}\n\nChanceux :\t{}".format(parfaits, chanceux)
print(msg)
```

```
--------------- Nombres remarquables dans [2 .. 1000] ----------------

Parfaits :      [6, 28, 496]

Chanceux :      [2, 3, 5, 11, 17, 41]
```

Exercice 10 : modules

1. Écrire un module de calcul des racines du trinôme réel : $ax^2 + bx + c$.
Le module définit une fonction trinôme avec les trois paramètres du trinôme, *a*, *b* et *c*. La fonction doit retourner un tuple dont le premier élément est le nombre de racines du trinôme (0, 1 ou 2), et les autres éléments sont les racines éventuelles.
Testez votre fonction avec les trois jeux de valeurs suivantes : 1,-3,2 1,-2,1 et 1,1,1.

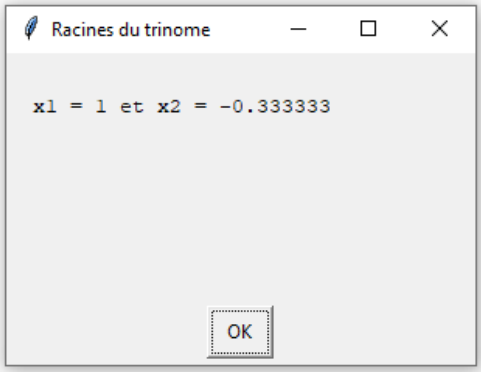
```
# Créé par Claude, le 17/09/2025 en Python 3.7
# fonction trinome pour résolution equation second degré
# import
from math import sqrt
# fonction
def trinome(a, b, c):
    delta = b**2 - 4*a*c
    if delta > 0.0:
        racine_delta = sqrt(delta)
        return (2, (-b-racine_delta)/(2*a), (-b+racine_delta)/(2*a))
    elif delta < 0.0:
        return (0,)
    else:
        return (1, (-b/(2*a)))
if __name__ == "__main__":
    print(trinome(1.0, -3.0, 2.0))
    print(trinome(1.0, -2.0, 1.0))
    print(trinome(1.0, 1.0, 1.0))
```

```
(2, 1.0, 2.0)
(1, 1.0)
(0,)
```

2. Écrire un programme principal utilisant le module précédent.
Les trois paramètres seront saisis et les résultats seront affichés dans une `messagebox` du module `TKINTER`.

```
# Créé par Claude, le 17/09/2025 en Python 3.7
# fonction trinome pour résolution equation second degré
# import
from Ex10 import trinome
from tkinter import messagebox
# programme principal ------------------------------------------
titre = "Resolution du trinome a*x**2 + b*x + c = 0"
a = float(input("a : "))
b = float(input("b : "))
c = float(input("c : "))
solutions = trinome(a, b, c)
if solutions[0] == 2:
    msg = "x1 = {:g} et x2 = {:g}".format(solutions[1], solutions[2])
elif solutions[0] == 1:
    msg = "x1 = x2 = {:g}".format(solutions[1])
else:
    msg = "Pas de racine reelle."
#print(msg)
messagebox.showinfo("Racines du trinome",msg)
```

a= 1, b= -3, c= 2



Exercice 11 : orienté objet

1. Définir une classe MaClasse possédant les attributs suivants :
données : deux attributs de classes : x = 23 et y = x + 5.
méthode : une méthode affiche contenant un attribut d'instance z = 42 et les affichages de y et de z.
Dans le programme principal, instanciez un objet de la classe MaClasse et invoquez la méthode affiche.

2. Définir une classe Vecteur2D avec un constructeur fournissant les coordonnées par défaut d'un vecteur du plan (par exemple : x = 0 et y = 0).
Dans le programme principal, instanciez un Vecteur2D sans paramètre, un Vecteur2D avec ses deux paramètres, et affichez-les.

3. Enrichissez la classe Vecteur2D précédente en lui ajoutant une méthode d'affichage et une méthode de surcharge d'addition de deux vecteurs du plan.
Dans le programme principal, instanciez deux Vecteur2D, affichez-les et affichez leur somme.

Ex11

```python
# Créé par Claude, le 23/09/2025 en Python 3.7
# classe
class MaClasse:
    """Definition d'une classe."""
    x = 23
    y = x + 5 # x et y : attributs de classe
    def affiche(self):
        """Definition d'une methode."""
        self.z = 42 # attribut d'instance (i.e. de l'objet self)
        print(MaClasse.y, end=" ") # dans une methode, on qualifie un attribut de classe...
        print(self.z) # ...mais pas un attribut d'instance
# programme principal -------------------------------------------------
obj = MaClasse()
obj.affiche() # a l'appel, self = obj. Idem : C.affiche(obj)
```

Ex11-2

```
class Vecteur2D:
    """Les Vecteurs plans."""
    def __init__(self, x0=0, y0=0):
        """Constructeur avec parametres par defaut."""
        self.x = x0 # initialisation de x et y, attributs d'instance
        self.y = y0
# programme principal -----------------------------------------------
print(" une instance par defaut ".center(50, '-'))
v1 = Vecteur2D()
print("x = %g, y = %g" % (v1.x, v1.y))
print()
print(" une instance par defaut ".center(50, '-'))
v1 = Vecteur2D()
print("x = %g, y = %g" % (v1.x, v1.y))
print()
print(" une instance initialisee ".center(50, '-'))
v2 = Vecteur2D(-5.2, 4.1)
print("x = %g, y = %g" % (v2.x, v2.y))
```

Ex11-3

```
# Créé par Claude, le 23/09/2025 en Python 3.7
class Vecteur2D:
    """Definition d'une classe."""
    def __init__(self, x0=0, y0=0):
        """Constructeur avec parametres par defaut."""
        self.x = x0 # initialisation de x et y, attributs d'instance
        self.y = y0
    def __add__(self, autre):
        """Addition vectorielle."""
        return Vecteur2D(self.x+autre.x, self.y+autre.y)
    def __str__(self):
        """Affichage d'un Vecteur2D."""
        return "Vecteur({:g}, {:g})".format(self.x, self.y)
# programme principal -----------------------------------------------
v1, v2 = Vecteur2D(1.2, 2.3), Vecteur2D(3.4, 4.5)
print(v1)
print(v2)
print(v1 + v2)
```

Afficher les nombres impairs dans une liste

```
# Créé par claude, le 30/09/2025 en Python 3.7
nombres = [1, 2, 3, 4, 5, 6, 7, 8, 9]
for nombre in nombres:
    if nombre % 2 != 0:
        print(nombre, end=" ")
```

Conversion décimale binaire

- l'opérateur **//** donne le quotient de la division euclidienne : **5 // 2** donne **2**;
- l'opérateur **%** donne le reste de la division euclidienne : **5 % 2** donne **1** ;
- **append** est une méthode qui ajoute un élément à une liste existante.
  Soit **T=[5,2,4]**, alors **T.append(10)** ajoute **10** à la liste **T**. Ainsi, **T** devient **[5,2,4,10]**
- **reverse** est une méthode qui renverse les éléments d'une liste.
  Soit **T=[5,2,4,10]**. Après **T.reverse()**, la liste devient **[10,4,2,5]**

On remarquera qu'on récupère la représentation binaire d'un entier **n** en partant de la gauche en appliquant successivement les instructions :

b = n % 2
n = n // 2